On Modeling Software Engineering Models' Intentions with Institutions

Mohamed Bettaz, Mourad Maouche

Philadelphia University

September 5, 2017

Motivation

- Know more on the concepts of modeling and model transformation
- Contribute to the effort conducted in the domain

Introduction

- Models [1]
  - "Nobody can just define what a model is, and expect that other people will accept this definition; endless discussions have proven that there is <u>no consistent common understanding</u> of models"
  - "A model is a simplification of a system built with an <u>intended</u> goal in mind. The model should be able to answer questions in place of the actual system"

[1] Source: Citation by P-A. Muller et al. , in the paper Modeling Modeling Modeling, Software and System Modeling, vol. 11, Springer, pp.347-359, 2012

Introduction (cont'd.)

- Model intention, intentional modeling [1] "... an understanding that reveals the "whys" behind the "whats" and the "hows"
  - Typically, process performers need models that detail the "hows",
  - Process managers prefer models that highlight the "whats", while
  - Process engineers charged with improving and redesigning processes need models that explicitly deal with the "whys"
- "Intentional modeling focuses on intentions and motivations of software systems... " [2]

[1] E.S.K. Yu et al., Understanding Why in Software Modeling Process Modeling, analysis and Design, in proc. of the 16th International Conference on Software Engineering, 1994
[2] J.C. Nwokeji et al., A Proposal for Consolidated Intentional Modeling Language, in proc. of the second workshop on graphical modeling language development, 2013

- In the context of intentional modeling, intention sharing is relevant to
  - Model transformation, and to the
  - Consistency of such a transformation

Overview

- Background & Related Work
- Objective
- Modeling the Intention
- An institutional Approach
- Concluding Remarks

Background

- A category theoretic approach [1]

$$X \xrightarrow[\mu]{} Y$$

Figure : A category theoretic answer: X is a representation of Y

- Models are called things

- Avoids many of the questions related to the debate on modeling (engineering models vs. simulation modeling for instance)

[1] P.A. Muller, F. Fondement, B. Baudry and B. Combemale, Modeling Modeling Modeling, Software and System Modeling, vol. 11, Springer, pp. 347-359, 2012

Background (cont'd.)

- For the authors of [1], an intention should be seen as mixture of requirements, behavior, properties, and constraints, either satisfied or maintained by the thing.

[1] P.A. Muller, F. Fondement, B. Baudry and B. Combemale, Modeling Modeling Modeling, Software and System Modeling, vol. 11, Springer, pp. 347-359, 2012

Background (cont'd.)

Relations between things and their intentions

Intention	Description	Notation
I(X) = I(Y)	X and Y have the same intention. They can represent each other.	$X \xrightarrow[\mu]{} Y$
$I(X) \cap I(Y) = \emptyset$	X and Y have totally different intentions.	$X \underset{\mu}{\dashrightarrow} Y$
$I(X) \cap I(Y) \neq \emptyset$	X and Y share some intention. X and Y can be partially represented by each other.	$X \underset{\mu}{\rightsquigarrow} Y$
$I(X) \subseteq I(Y)$	The intention of X is part of the intention of Y. Y can be partially represented by X.	$X \xrightarrow[\mu]{}\mid Y$
$I(X) \supseteq I(Y)$	X covers the intention of Y. X can represent Y.	$X \xmapsto[\mu]{} Y$

Figure : Things and intentions

Background (cont'd.)

- From the figure (set theoretic representation) and the table:
  - In models' representations there is clear separation between the essence of a model and its intention
  - Sharing of intentions doesn't address consistency
  - Various arrows are used to represent relations between intentions

Objective

- Use institutions to
  - Characterize the concept of intention
  - Express relations between intentions in terms of relations between institutions (comorphisms, morphisms, *spans* of comorphisms, *co-spans* of comoprphisms, etc.)

    For instance semi-comorphisms might be used to express what is called *shift in intentions*

- In other words adopt an institutional approach (instead of the languages' approach or the categorical approach)

- <u>Illustration:</u>
  - Software engineering models represented as usual, where
  - Intention is seen as a set of constraints

Background (cont'd.)

Software engineering models (SE-models according to [1])

- Different system views are usually described by different (sub-) models
- In the context of MDA, various acronyms are used
  - CIM (Computation Independent Models) to address the domain or requirement viewpoint,
  - PIM (Platform Independent Models) to represent the design viewpoint, and
  - PSM (Platform Specific Models) to deal with the implementation viewpoint
- Models' transformations (MDE methodology)
  - CIM-PIM, PIM-PIM, PIM-PSM, and others

[1] A. Boronat, A. Knapp, J. Meseguer, and M. Virsing, What is a Multi-Modeling Language, LNCS, vol. 5486, 2009

Related Work

P.A. Muller, F. Fondement, B. Baudry and B. Combemale, Modeling Modeling Modeling, Software and System Modeling, Springer, 2012

A. Boronat, A. Knapp, J. Meseguer, and M. Virsing, What is a Multi-Modeling Language, Springer, 2009

M.V. Cengarle and A. Knapp, An Institution for UML 2.0 Static Structures, Technical Report TUM-10807, Technische Universitat Munchen, 2008

M.V. Cengarle, A. Knapp, A. Tarlecki and M. Virsing, A Heterogeneous Approach to UML Semantics, Springer, 2008

J.C. Nwokeji, T. Clark and B.S. Barn, A Proposal for Consolidated Intentional Modeling Language, in Proc. of the Second Workshop on Graphical Modeling Langauge Development, 2013

H. Baumeister, M. Bettaz, M. Maouche, and M. Mosteghanemi, Springer, 2015

Characterizing the intention

Could be done using various approaches (depending on the formalism used to describe the model view)

We see at least two approaches:

- Through a comorphism between institutions
- Through a morphism of specifications over an institution

Characterizing the intention (through specification morphism)

- Specification and specification morphism:
  - A specification $Sp = (\Sigma, \Phi)$ over an institution $\mathcal{I}$ consists of a signature $\Sigma \in Sig$ and a collection of Σ-sentences $\Phi \subseteq Sen(\Sigma)$
  - A specification morphism $\phi : (\Sigma, \Phi) \to (\Sigma', \Phi')$ is a signature morphism $\phi : \Sigma \to \Sigma'$ preserving the truth of sentences, i.e., $\Phi' \models_{\Sigma'} Sen(\Phi)$ (a specification morphism is a signature morphism where each sentence of the first specification maps to a theorem of the second specification)

- Specification morphism is induced by inheritance in institutions
- Inheritance is a general construction that works in any institution $\mathcal{I}$

Using inheritance in Object-Z [1]

Definition

Let $Sp_1 = (\Sigma_1, \Phi_1)$ be a specification in $Spec_{\mathcal{I}}$ and ϕ a signature morphism from Σ_1 to Σ_2. Let Φ be a set of sentences over Σ_2. Then $Sp_3 = (\Sigma_2, Sen(\phi)(\Phi_1) \cup \Phi)$ is a specification created by inheriting the specification Sp_1 via ϕ.

Lemma

If Sp_3 is a specification created by inheriting from specification Sp_1 via ϕ, then ϕ is a specification morphism from Sp_1 to Sp_3.

$Sp_2 = (\Sigma_2, \Phi)$ might be used to capture the intention

[1] H. Baumeister, M. Bettaz, M. Maouche, and M. Mosteghanemi, an Institution for Object-Z with Inheritance and Polymorphism, LNCS, 2015.

A running example: Train control system

- The domain of interest deals with trains, transporting of passengers, platforms, and moving of trains from platforms to platforms
- Trains may be stopped either at platforms or outside of them
    - No more than one train might be stopped at a given platform at a given time

Train control system: Modeled views

- Two views: CIM and PIM
  - CIM: UML class diagram and OCL notation
  - PIM: Object-Z specification
- UML: a *standard* choice for addressing the requirement level
- Object-Z:
  - *Proximity* with associations integrating UML and OCL,
  - Ability to refine such associations in a straightforward way
- We give example models for both views, with
- Focus on the PIM (show how to capture the intention)

The CIM view

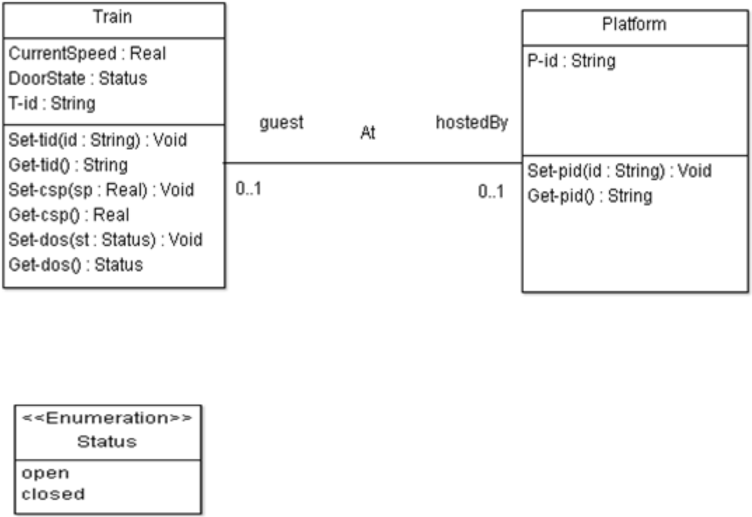

Figure : CIM viewpoint represented by a UML class diagram

The CIM intention

$$context : t : Train$$
$$inv : t.CurrentSpeed = 0 \text{ } or \text{ } t.DoorState = closed$$
$$context : t : Train$$
$$inv : t.hostedBy \neq \emptyset \text{ } or \text{ } t.DoorState = closed$$

The PIM view

- The PIM (as a design model) specifies the what, in conformity with the why (expressed as a CIM intention), i.e.,
  - CIM intention is part of the PIM view
- We use a separation of concern modeling strategy (implemented using inheritance in Object-Z)
  - Distinguishing between the *essence* of the model and the intention

The PIM view: Implementation of the CIM class diagram

- UML CD classes are refined into a corresponding Object-Z classes
- The UML CD association is represented by reference attributes
- The UML CD is represented by an Object-Z system class

The PIM view: Implementation of the CIM class diagram (cont'd.)

Train

$T - id : Sring$
$DoorStatus : Status$
$CurrentSpeed : \mathbb{R}$
$hostedBy : \mathbb{P}\ Platform$

$\#hostedBy \leq 1$

INIT

$DoorState = closed$
$CurrentSpeed = 0$
$hostedBy \neq \varnothing$

Set − tid

$\Delta(T - id)$
$id? : String$

$T - id' = id?$

Get − tid

$id! : String$

$id! = T - id$

....

Figure : Train Object-Z class

The PIM view: Implementation of the CIM class diagram (cont'd.)

Platform

$P - id : Sring$
$guest : \mathbb{P}\ Train$

$\# guest \leq 1$

INIT

$guest = \varnothing$

Set − pid

$\Delta(P - id)$
$id? : String$

$P - id' = id?$

Get − pid

$id! : String$

$id! = P - id$

Set − guest

$\Delta(guest)$
$gs? : \mathbb{P}\ Train$

$\# gs? \leq 1$
$guest = \varnothing$
$guest' = gs?$

Get − guest

$gs! : \mathbb{P}\ Train$

$gs! = guest$

The PIM view: Implementation of the CIM class diagram (cont'd.)

TrainSystem

$trs : \mathbb{P}\ Train$
$pfs : \mathbb{P}\ Platform$

$$\forall\, t : trs \bullet (t.hostedBy = \varnothing) \lor \exists\, pf : pfs \bullet (t.hostedBy = \{pf\})$$
$$\forall\, pf : pfs \bullet (pf.guest = \varnothing) \lor \exists\, t : trs \bullet (pf.guest = \{t\})$$

Figure : Train system Object-Z class

The PIM view: Implementation of the CIM intention

$$\textit{ControlledTrain}$$

Train

Move

$\Delta(CurrentSpeed)$
$sp? : Real$

$CurrentSpeed = 0$
$sp? > 0$
$DoorState = closed$
$CurrentSpeed' = sp?$

Stop

$\Delta(CurrentSpeed)$

$CurrentSpeed' = 0$

OpenDoors

$\Delta(DoorState)$

$DoorState = closed$
$CurrentSpeed = 0$
$hostedBy \neq \varnothing$
$DoorState' = open$

CloseDoors

$\Delta(DoorState)$

$DoorState = open$
$CurrentSpeed = 0$
$hostedBy \neq \varnothing$
$DoorState' = closed$

The PIM view: Implementation of the CIM intention (cont'd.)

AbstractSystem

TrainSystem

$ctrl : \mathbb{P}\, ControlledTrain$

$\#ctrl = \#trs$

Figure : AbstractSystem Object-Z class

Object-Z institution [1]

An Object-Z signature Σ is a triple $\Sigma = S \cup F \cup \pi$, where

- $S = C \cup T \cup P$ is a set of class names (C), type names (T), and polymorphic class names π
- $F = B \cup R \cup O$ is a set of operations representing basic attributes (B), reference attributes (R), and operation schemas (O)

[1] H. Baumeister, M. Bettaz, M. Maouche, and M. Mosteghanemi, an Institution for Object-Z with Inheritance and Polymorphism, LNCS, 2015.

PIM signature

- C = {Train, Platform, TrainSystem}
- T = {String, Real, Status}
- P = $\emptyset$ (there are no polymorphic classes in our example)
- B= $B_{Train \to String} \cup B_{Train \to Status} \cup B_{Train \to Real} \cup B_{Platform \to String}$ where,
  - $B_{Train \to String} = \{\text{T-id}\}$

    ...
- R= $R_{Train \to \mathbb{P} Platform} \cup R_{Platform \to \mathbb{P} Train} \cup R_{TrainSystem \to \mathbb{P} Train} \cup R_{TrainSystem \to \mathbb{P} Platform}$ where,
  - $R_{Train \to \mathbb{P} Platform} = \{\text{hostedBy}\}$

    ...
- O= $O_{Train, <String>} \cup O_{Train, <Real>} \cup O_{Train, <Status>} \cup O_{Train, <\mathbb{P} Platform>} \cup O_{Platform, <String>} \cup O_{Platform, <\mathbb{P} Train>} \cup O_{Train, <>, String} \cup O_{Train, <>, Status} \cup O_{Train, <>, Real} \cup O_{Train, <>, \mathbb{P} Platform} \cup O_{Platform, <>, String} \cup O_{Platform, <>, \mathbb{P} Train}$ where,
  - $O_{Train, <String>} = \{\text{Set-id}\}$

    ...

Object-Z Sentences

Three kinds of sentences:

- For initial state schemas, they have the form: $Init_c : P$
- For state schemas' invariants, they have the form: $Inv_c : P$
- For operation schemata, they have the form: $o(id? : id, ..., id? : id, id! : id)[id, ..., id] : P$

Object-Z institution (cont'd.)

PIM Sentences

- $Init_{Train}$: $DoorState = closed \wedge CurrentSpeed = 0 \wedge hostedBy \neq \emptyset$
- $Init_{Platform}$: $guest \neq \emptyset$
- Inv_{Train} : $\forall t : Train.\#(t.hostedBy) \leq 1$
- $Inv_{TrainSystem}$: $(\forall t : trs.(t.hostedBy = \emptyset)) \vee (\exists pf : pfs \bullet t.hostedBy = \{pf\})$
- $\cdots$
- $Set - id(id? : String)[T - id] : T - id' = id?$
- $\cdots$

The PIM view: Implementation of the CIM intention (cont'd.)

- Specification of controlled train
  - $Sp_{ControlledTrain} = (\Sigma_{ControlledTrain}, Sen_{OZ}(\phi)\Phi_{Train} \cup \Phi_{ControlledTrain})$ where
  - $\phi : \Sigma_{Train} \rightarrow \Sigma_{ControlledTrain}$ is a signature morphism mapping *Train* to *ControlledTrain* (identity on attributes and operations)

- The intention is given by $(\Sigma_{ControlledTrain}, \Phi_{ControlledTrain})$

The PIM view: Implementation of the CIM intention (cont'd.)

The intention $(\Sigma_{ControlledTrain}, \Phi_{ControlledTrain})$
$\Sigma_{ControlledTrain} = \{ControlledTrain, String, Real, Status\}$
$\Phi_{ControlledTrain} = \{$

- $Move(sp? : Real)[CurrentSpeed] : CurrentSpeed = 0 \; and \; DoorState = closed \; and \; sp? > 0 \; and \; CurrentSpeed' = sp?$
- $Stop()[CurrentSpeed] : CurrentSpeed > 0 \; and \; CurrentSpeed' = 0$
- $OpenDoors()[DoorState] : CurrentSpeed = 0 \; and \; hostedBy \neq \emptyset \; and \; DoorState' = open$
- $CloseDoors()[DoorState] : CurrentSpeed = 0 \; and \; hostedBy \neq \emptyset \; and \; DoorState' = closed$

  $\}$

The PIM view: Implementation of the CIM intention (cont'd.)

- Specification of the PIM
  - $Sp_{PIM} = Sp_{Train} \cup Sp_{Platform} \cup Sp_{ControlledTrain} \cup Sp_{TrainSystem} \cup Sp_{AbstractSystem}$
  - $\phi : \Sigma_{Train} \rightarrow \Sigma_{ControlledTrain}$ is a signature morphism mapping *Train* to *ControlledTrain* (identity on attributes and operations)
- The intention is given by $(\Sigma_{ControlledTrain}, \Phi_{ControlledTrain} \cup \Phi_{AbstractSystem})$

Concluding remarks

- Refinement of the intention (specific system)
- Characterization of the CIM intention (at the CIM level): embedded morphism between UML static structures' institution and OCL institutution (done: not presented here)
- Mapping of intentions (worked-out intuitively : not presented here)
- Future:
  - Characterize intention at the CIM level by specification morphism in OCL institution
  - Formalize mapping of intentions (OCL, Object-Z)